FEATURE

Developing the GX Graphics Accelerator Architecture

High-level graphics on entry-level workstations become practical with this new approach to acceleration.

Curtis R. Priem

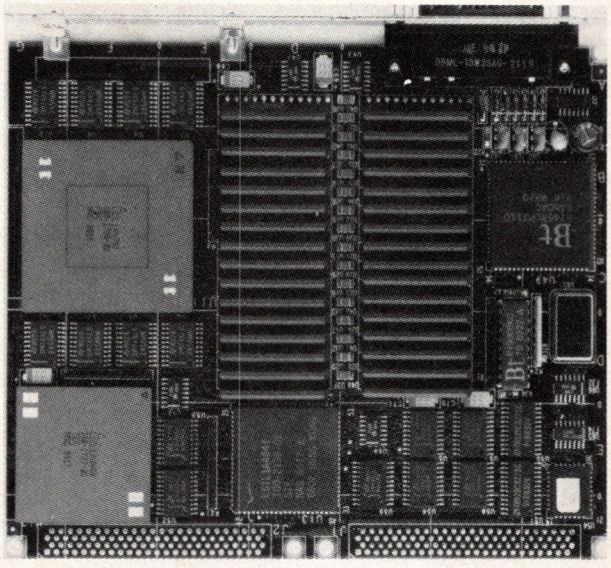
Sun Microsystems, Inc.

Sun's GX Graphics Accelerator Board (shown in Figure 1 on the next page) represents a significant departure from traditional graphics acceleration. Previous graphics accelerators usually contained some intelligence, either a CPU or bit-slice processor, which controlled the specialized graphics hardware and processed much of the rendering calculations. The support hardware, memory, cache, power supply, and board size prevented, until now, graphics acceleration from being placed on entry-level workstations. In the GX, however, the host CPU functions as the intelligent controller and two large ASICs (application-specific ICs) supply hardwired graphics functions.

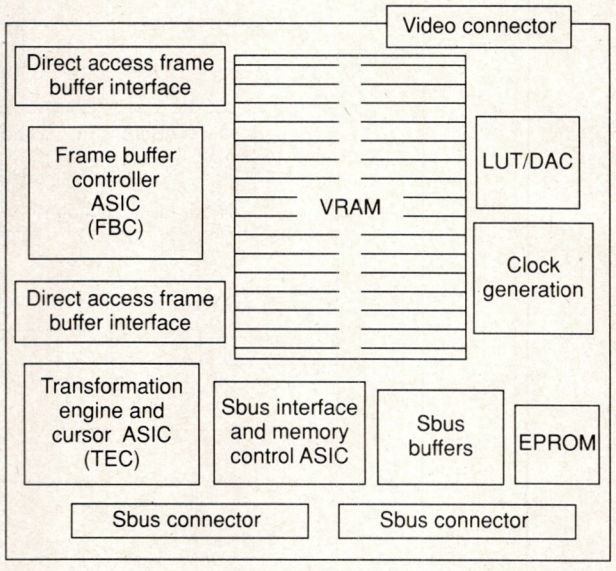
The 80/20 rule and graphics

We determined the GX's functionality based on the dollar size of its target markets. We applied a variant of the 80/20 rule popularized in RISC (reduced instruction-set computer) designs to graphics algorithms. This method identifies the 20 percent of CISC (complex instruction-set computer) instructions that are used 80 percent of the time. These become the instructions for the RISC microprocessors. The 80/20 rule, as it applies to the GX, meant identifying the 20 percent of the graphics algorithms that would have the biggest impact on at least 80 percent of the dollars within each targeted market sector.

0272-1732/90/0200-0044$01.00 © 1990 IEEE



(a)



(b)

Figure 1. The GX Sbus Graphics Accelerator Board (a) and the layout diagram (b).

An arbitrary quadrilateral is the GX's only geometric primitive. The GX directly supports the most commonly used graphic drawing primitives (points, lines, triangles, rectangles) as degenerate cases of a quadrilateral. The operations and primitives that the GX does not directly support are generally available through multiple applications of one (or more) of the supported operations. This approach resembles the way CISC instructions can be made up by combining simpler RISC instructions.

The GX does not support circle and arc primitives. However, it can readily approximate them with short line segments. In addition, the GX supports flat shading of images only when every pixel on a polygon is the same color or intensity. However, by breaking the object into many smaller objects (tessellation) each with its own color, users can obtain a visually acceptable approximation to Gouraud-shading techniques.[1,2] The high performance of the GX's rendering hardware makes this approach attractive. Going one step further, when a tessellated polygon gets close to one pixel in size, the result becomes the Phong-shading technique.

The GX accepts and entirely completes its low-level graphics operations without host assistance. However, the GX does not control nor drive the graphics application; the host CPU continues these functions. The host executes the application until it encounters a function supported by the GX (directly or indirectly by compositing multiple supported operations). At this point, the CPU writes parameters of the function to the GX's memory-mapped data registers. The GX first validates these parameters prior to the start of any operation. The CPU reads a status register to see if any unsupported or errant condition exists. If one does,[3] the host CPU handles the exception in software.

Integration

High-level integration was one of the key ideas behind the GX concept. In fact, the code name for the GX during its development was LEGO, which stood for Low End Graphics Option. "Low end" did not describe performance; it described the targeted workstations. The GX board (including frame buffer) requires less than 20 watts from the workstation's power supply.

In previous graphics accelerators the performance was proportional to its price. With large graphics accelerators the user must pay for a large power supply, extra cooling, and a large chassis. These added costs alone have kept high-performance graphics from appearing on desktop computers.

To achieve GX suitability for the low-end workstation, we chose to integrate the GX into two large ASICs,[4] the Frame Buffer Controller and the Transformation Engine and Cursor chip. The FBC acts as the graphics-rendering engine. It contains 43,000 gates (170,000 transistors) packaged in a 223-pin grid array and uses LSI Logic's 1.5-μm sea-of-gates technology. The TEC transforms all the graphics, provides a hardware cursor, and contains the timing generation logic for the frame buffer. It contains 25,500 gates, over

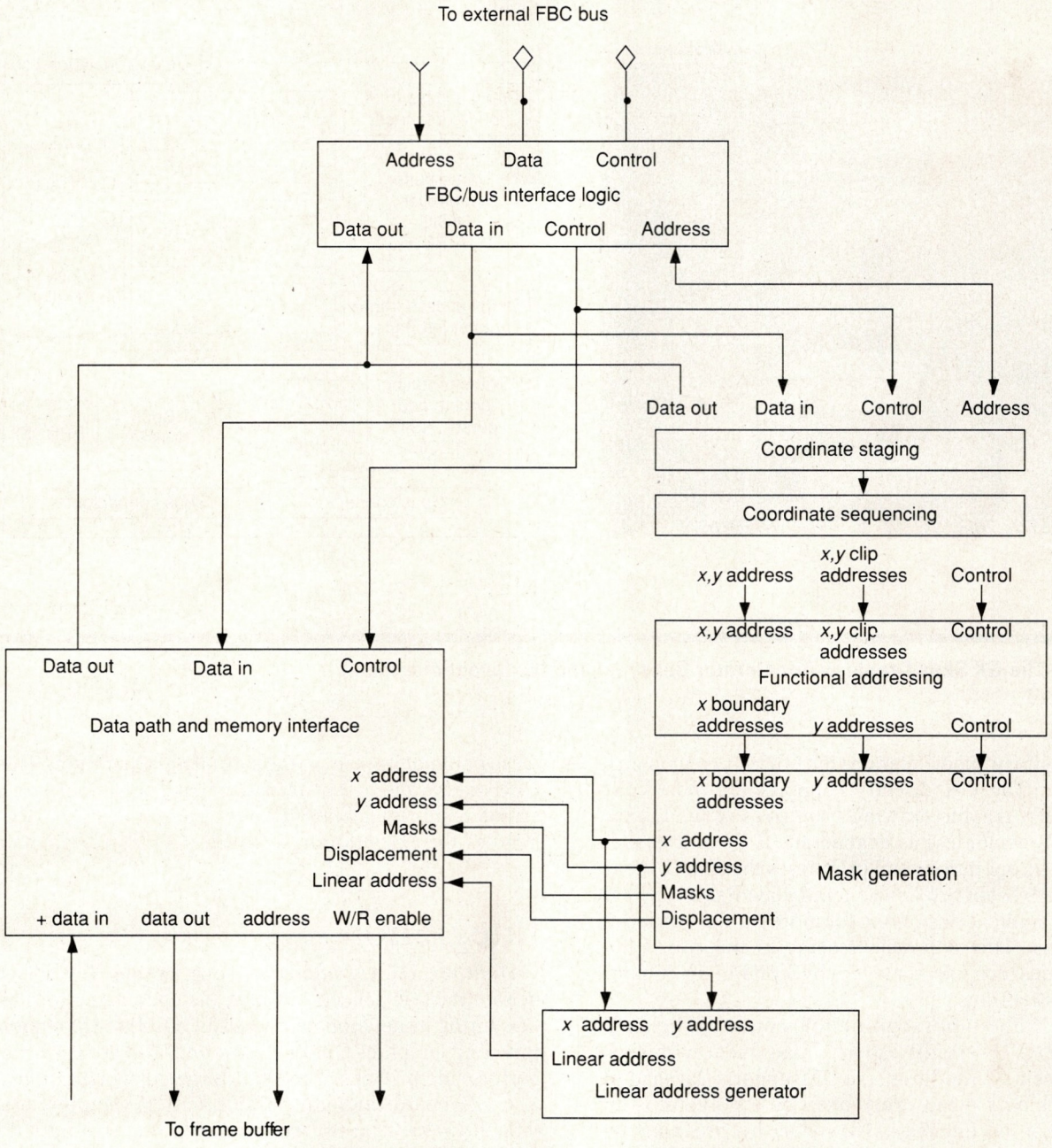


Figure 2. FBC block diagram.

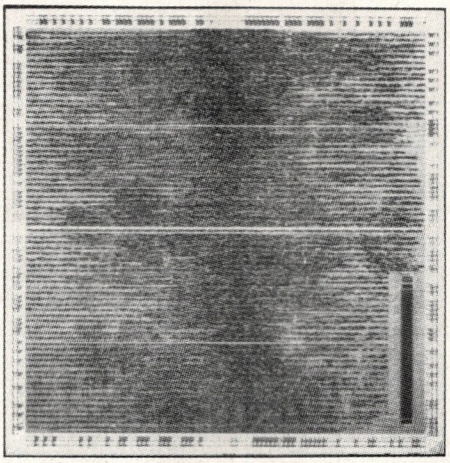
4 Kbits of RAM (212,000 transistors) in a 95-PGA package, and uses LSI Logic's 1.5-µm standard-cell technology.

Frame Buffer Controller chip

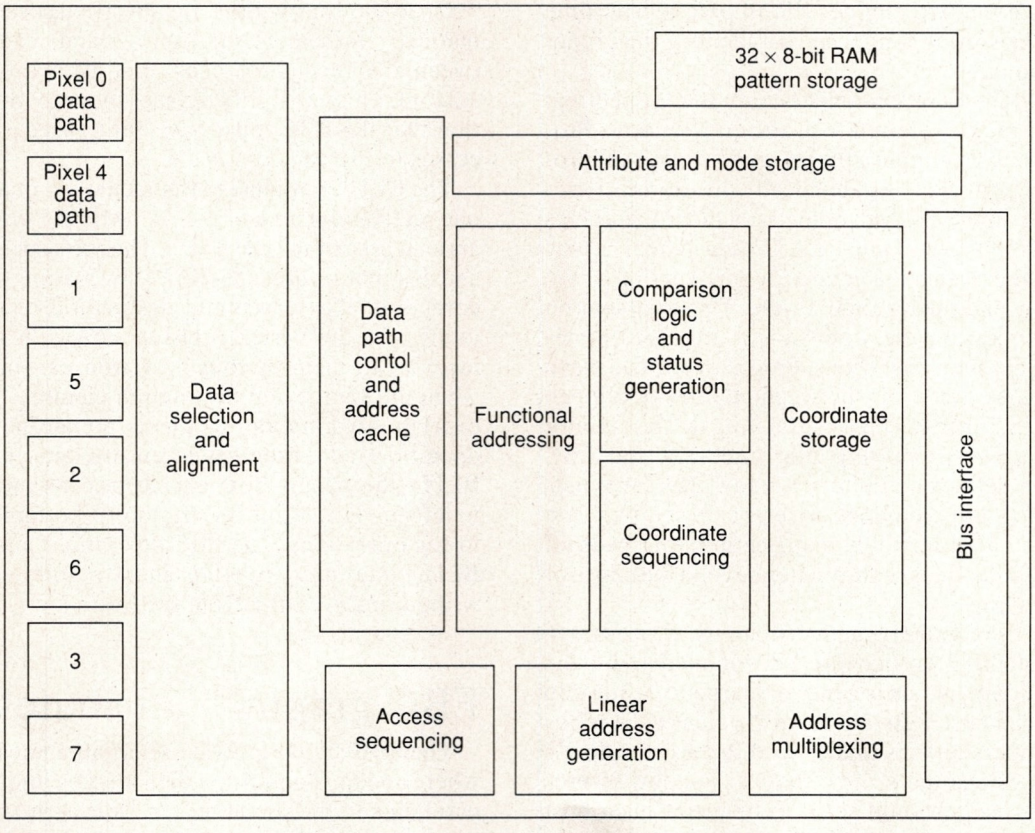
The FBC chip detailed in Figures 2 and 3 (on the next page) performs all rendering or drawing operations and places the results in the frame buffer. It processes only three instructions:

- DRAWs, which render arbitrary, filled quadrilaterals,
- FONTs, which display precomputed pixel images, and
- BLITs, which transfer block images.

The first selected rendering algorithm draws only arbitrary filled quadrilaterals that can be self-intersecting and degenerate. Due to limited silicon resources, we felt it very important to simplify geometric rendering into one algorithm in the FBC.



(a)



(b)

Figure 3. The FBC die (a) and its layout (b). (Die photomicrograph courtesy of LSI Logic.)

Since it processes degenerate quadrilaterals (repeated vertices), the FBC can render points, lines, and triangles.

To define each quadrilateral to be drawn, four (x,y) coordinate pairs must first be loaded into the FBC. Special addressing modes allow a point, line, triangle, or rectangle to be completely defined by loading the minimal number of vertices. The chip can express a point as a degenerate quadrilateral in which all of the vertices are the same. In this case, the chip's addressing

logic replicates the one vertex into all four vertex registers. In a similar manner, the programmer can specify a line with just two vertices, a triangle with three, and an aligned rectangle with two opposite corner vertices. The FBC also supports chained addressing, which avoids reentering common vertices in polylines, triangle/quadrilateral strips, and meshes.

The FBC renders the object defined in its registers using a variant of the Bresenham line-drawing algorithm.[5-8] Internal to the FBC, two Bresenham algorithm engines run in parallel, each processing one edge of the arbitrary quadrilateral. Additional logic identifies the left and right edges for each scan line, sorts them, and applies clipping calculations. The scan line then fills between these two points with data from a pattern RAM. Degenerate quadrilaterals, such as points or lines, simply cause the two Bresenham engines to process the same edge. This duplication is of no consequence since the two engines run in parallel (no time lost), and the sorting logic can still identify the proper scan-line boundaries.

The drawing operation renders only into a specified rectangular region referred to as the clip window. Processed, invisible (clipped) pixels do not represent productive work in that no change to the frame buffer results. For most clipped objects, standard graphics software would first compute and then render a new object that represents the geometric intersection of the clip window and the original object. The computation of the vertices of the new object involves time-consuming division operations. Programmers can make a performance trade-off between deferring clipped object processing to the host or allowing the accelerator to handle it. In supporting this trade-off, the FBC architecture defines a clip test window that surrounds the specified clip window. Users can set this test window at a programmable distance in both x (edges) and y (horizontal lines) coordinates beyond the edges of the clip window.

The FBC completely and efficiently processes a clipped object if its vertices are all contained within the clip test window (which is generally larger than the clip window). The host software more efficiently (in terms of time) renders objects that exceed the test window's boundaries. Since the size of the test window is programmable, users should set it to optimize each host workstation.

The FBC will actually handle many objects that have vertices outside the test window when certain conditions are met. The flexible drawing-engine architecture allows an object to be rendered in either top-to-bottom or bottom-to-top order. The drawing engine renders an object whose left, right, and bottommost vertices appear inside the test window, with the uppermost vertex outside it, from bottom to top. It stops when it reaches the upper edge of the clip window. Thus, the pixels at the top of the object that exceed the clip window need never be processed.

The FBC also accelerates the rendering of precomputed raster images into the frame buffer. These images can exist in either a monochrome (1-bit-per-pixel) format (text) or a color (8-bit-per-pixel) format (image). The FBC accelerates the display of text and images by supporting x/y addressing, address autoincrementing, clipping, raster operations, and frame-buffer alignment. The FBC also provides 1-to-8 plane expansion for monochrome text or images.

The FBC determines the target location of a font by loading the horizontal left and right points of the scan line in which the font will be placed. This arrangement allows different-width fonts (in terms of pixels) to be written. By specifying the font width this way, versus providing a pixel-width value, the same clipping and address processing logic that handles quadrilaterals handles fonts. (The operation is similar to drawing textured horizontal lines.)

Once the font widths are specified, users can automatically increment the font's x and y addresses between the receipt of each font data access by any amount. This capability reduces the amount of information that the FBC must have for consecutive font accesses to just the data itself.

The BLIT command to transfer block images permits one rectangular area of the frame buffer to be copied to another area of the same size. The FBC specifies source and destination areas for BLITs by loading the diagonal corners of the source and destination rectangular regions into the vertex registers. As with any type of memory-to-memory copy operation, care must be taken when the source and destination elements of the copy overlap. Overlapping elements dictate how the copy operation must progress to ensure data integrity. For BLITs, this means the copy may be constrained to occur in left-to-right, right-to-left, top-to-bottom, or bottom-to-top operations. The FBC determines and executes a BLIT for arbitrary overlaps in the required progression without intervention from the host.

FBC graphics attributes

The FBC supports a rich set of attributes that define where and how new pixel data merges into the existing pixel data in the frame buffer. This approach is analogous to a CPU that supports a generic operate instruction in which the actual ALU operation to be performed is specified in some auxiliary register.

The architecture stipulates that all attributes are available for use and consistently applied across all of its three graphics commands. An example of a common attribute is color; the FBC supports both foreground and background color. Once established, the colors and other attributes will affect all subsequent operations.

The predominant mechanism for defining the way a new per-pixel value merges bit-wise with the existing value involves raster operations (Rops). The FBC's

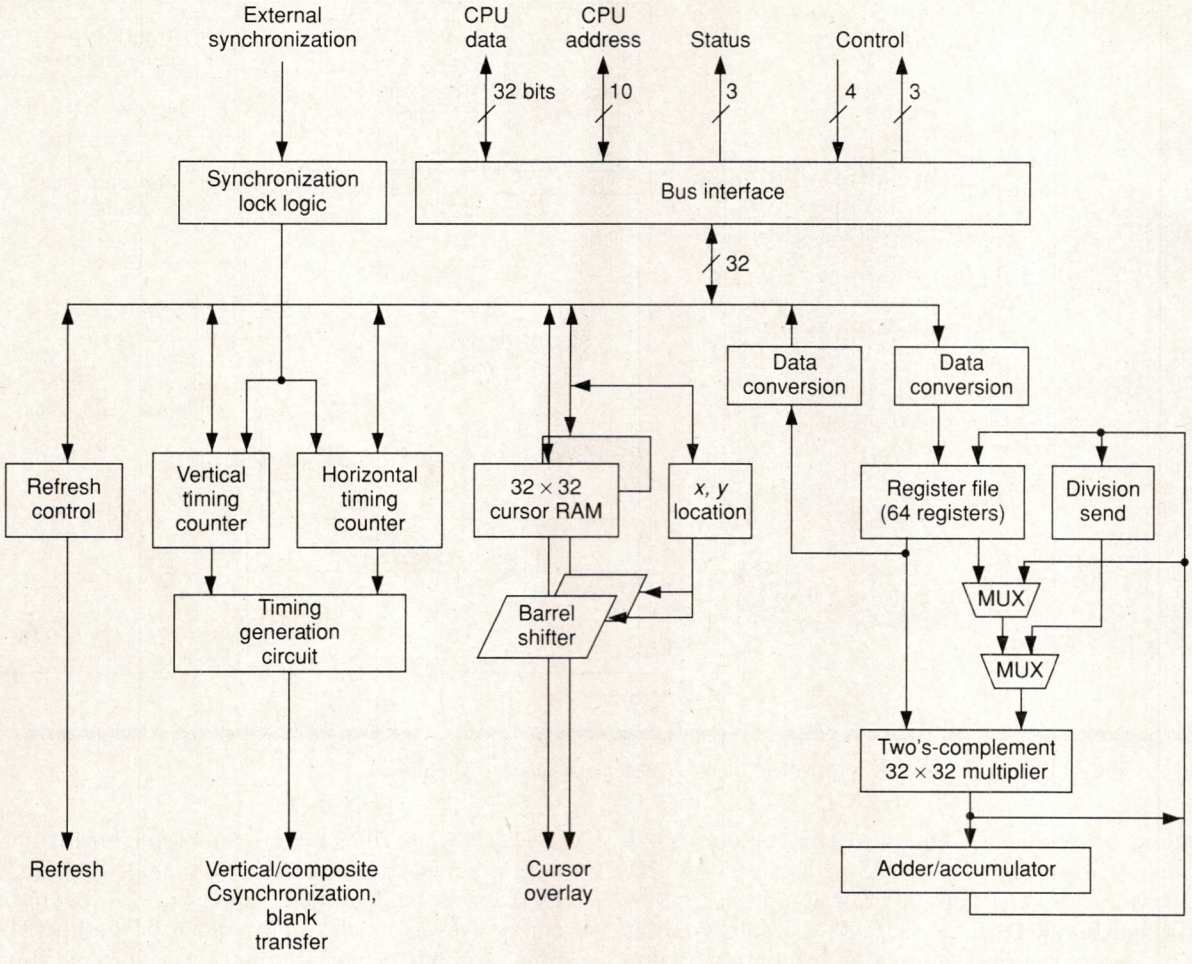


Figure 4. TEC block diagram.

Boolean Rop mode performs logical (And, Or, NAND, XOR, SRC, DEST) bit-wise operations between each new source pixel and the existing destination pixel at the target address of the frame buffer. The FBC supports all 16 Boolean Rops as defined in Sun's Pixrect graphics library. In addition, the FBC integrates the foreground and background colors into the final Rop determination, permitting 65,536 possible Boolean Rops.

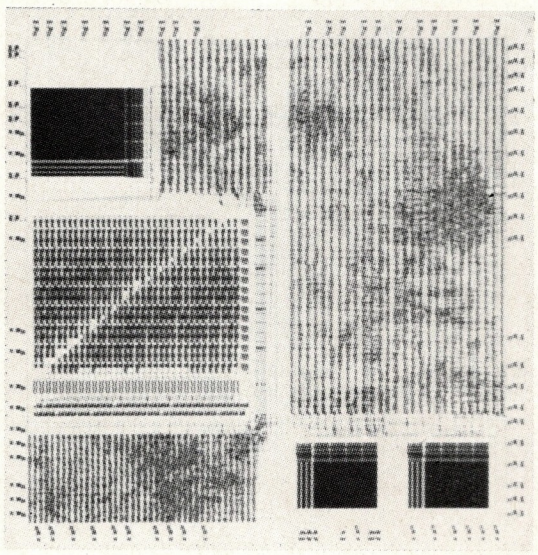
A plane-masking capability lets users select the bit planes within each pixel for modification. The current implementation of the FBC uses the write-per-bit capability of its video RAMs to restrict individual planes from being modified. Thus, each VRAM actually performs the plane-masking calculations so that no performance degradation occurs when users enable this option.

The FBC uses a pattern RAM activated by the Draw command. The 16 × 16, one-bit-deep (monochrome) pattern fills quadrilaterals. The pattern repeats across the frame buffer every 16 pixels and can be aligned to arbitrary offsets in both x and y directions. The FBC's color and Rop capabilities permit the monochrome pattern to expand into an 8-bit color index.

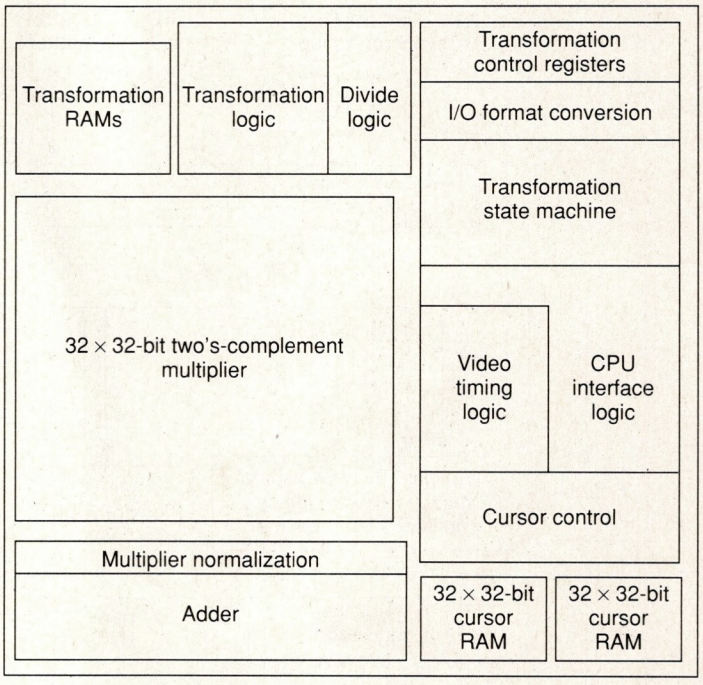
The TEC chip

The transformation engine and cursor chip modifies all graphics, supports the computational requirements of graphics programs,[2] provides a hardware cursor, and contains the timing generation logic for the frame buffer. (See Figures 4 and 5.) The TEC contains a 64-register file and a 32-bit, signed multiply/accumulator, which combine to provide arithmetic calculations at a rate of up to 50 million floating-point operations per second.

The TEC includes explicit support for the typical 2D and 3D graphics-transformation pipeline. The



(a)



(b)

Figure 5. The TEC die (a) and its layout (b). (Die photomicrograph courtesy of LSI Logic.)

modeling and viewing, or MV, matrix can consist of anything from a simple 3 × 2 matrix used by 2D graphics to a 4 × 4 matrix used by 3D graphics. The MV matrix accommodates rotation, scaling, and translation of an object or its viewing position. An optional clip-checking operation adjusts a rectangular region for 2D graphics, a viewing pyramid for 3D graphics, and a double pyramid for homogeneous graphics such as parametric curves.[9] The TEC also provides an optional perspective divide function that uses Newton-Raphson iterations.[10,11]

At the end of the pipeline, an optional matrix (the virtual device coordinate) transforms the coordinates into the final screen-coordinate space. The host can read and/or automatically load transformation results into the FBC for rendering. The host also has access to intermediate pipeline results.

The TEC's external interface supports three different data formats. Data can be read or written in either signed integer, fract (16.16), or IEEE-754 single-precision floating-point formats.[12] Different formats can be intermixed, and all conversions are pipelined to avoid a performance penalty. The TEC accepts both absolute and relative 2D (x,y), 3D (x,y,z), and homogeneous (x,y,z,w) coordinate data. Since the TEC's math capability extends to general-purpose vector operations, it calculates the arithmetic required for matrix concatenation, dot products, lighting models, and even some common imaging operations such as convolutions.

The TEC provides a three-color (and transparent), 32 × 32-pixel hardware cursor. We included this capability when we realized that support of a software-based cursor uses as much as 20 percent of the host CPU's time. So, with a workstation processing 5 million instructions per second, the CPU can devote as much as 1 MIPS to manipulating the cursor and its bitmap. With the TEC's hardware cursor, the CPU loads only a new (x,y) cursor coordinate into the hardware when the mouse moves. The three-color cursor accommodates images with an outline in the opposite color of its body. This capability prevents the cursor from ever being lost in a solid-colored background (as in a window border).

The TEC also contains the timing generators. A programmable timing generation circuit allows software to program the TEC for different-resolution monitors and different frequencies. The TEC generates the signals required for the VRAM's transfer and refresh cycles, and the synchronization and blanking signals required by the monitors.

Software in silicon

Even though the GX interface consists of only three simple instructions and appears very much like a RISC, internally the ASICs function in "super-CISC" style. The chips don't have the complex instructions found in CISC processors; rather they perform the equivalent of very complicated software subroutines.

The GX provides the functional equivalents of

10,000 lines of code. Neither the FBC nor the TEC ASICs contains actual software or microcode. The subroutines proceed instead in a massively parallel way, via state machines (the FBC contains 12 state machines) and combinatorial logic (the FBC's Rop logic and barrel shifters repeat eight times). Since we lost most resemblance to the original routines, we developed numerous new algorithms and have applied for 15 patents related to the FBC and six patents associated with the TEC.

This design approach meant ignoring the normal sequential way of implementing a subroutine or algorithm. For the GX, we used large amounts of combinatorial logic operating in parallel with many state machines that supported wide data paths. (The FBC's main data path is 128 bits wide.) We emphasized a steady, high data flow rate and did not limit the number of gates. We also provided numerous replicated resources just to enhance overall throughput.

Infinitely fast CPUs

As mentioned earlier, the most common way of designing a graphics system is to build a board that has some type of intelligent device on it. The device may be a microprocessor or a microcoded, custom bit-slice processor. Unfortunately, this device uses the same basic technology as the host CPU but is put together in a little different way to optimize the graphics function. When the next-generation technology comes along, the host CPU generally performs faster, quickly making the current-generation's graphics subsystem obsolete.

A good example of this phenomenon relates to the development of the original IBM Professional Graphics Adapter. PGA designers decided to use an 8088 processor to handle image-rendering operations for the frame buffer. This design worked well on the company's PC XT, which also had an 8088, and even on its AT, which had an 80286 processor. In these two cases, the multiprocessing arrangement was effective.

But upon adding the original PGA to a 80386-based PC, the overall system performance actually slowed down. The 80386 can both execute the application software and render the graphics in less time than the PGA could render just the graphics—not an effective multiprocessing setup. As things turned out, the 80386-based PCs were available within two years of the time the PGAs became available.

The rate of increase in CPU performance seems to be increasing with the advent of the new RISC CPUs. Following company cofounder Bill Joy's Law, we can expect CPU performance to double roughly every year [MIPS = 2 raised to (current year − 1984)]. In other words, designers would have to develop a new graphics board every year.

Instead, we wanted to come up with an architecture that would last for several years. We knew that meant that the GX graphics accelerator could not become the bottleneck of the system if a 100-MIPS CPU was connected to it. The GX could not perform slower than the worst case in which an infinitely fast CPU was connected directly to the frame buffer.

With such a CPU connected to a frame buffer, the VRAMs become the bottleneck of the system. We knew that if we built our graphics accelerator, the worst possible thing that could happen would be that 10 years from now the CPU would render as fast as the accelerator.

The bottleneck would occur at a different location in our design. A common bottleneck for previous accelerators had been determined by the transformation speed that could be achieved using the chosen technology. Designers then developed the rendering section to draw polygons only at the rate that transformed coordinates could be delivered.

Table 1 (on next page) lists the device and related frame-buffer bandwidths for a number of common VRAM organizations. We assumed a one-megapixel, or 1,024 × 1,024 × 8, frame buffer. Notice that even though the bandwidth of each VRAM generally increases, the time required to load all new data into the VRAM lengthens, and the bandwidth into the frame buffer decreases. This phenomenon is currently unique to graphics operations. In these operations the size of the frame buffers does not increase with time due mainly to the cost of higher resolution monitors. Workstations may start to run into this problem in a few years unless the operating systems, windowing systems, and applications continue to increase in size proportionally to dynamic-RAM density. Since the interface to the VRAM is not going to double in performance every year, we achieve a very stable interface for designs.

Scalable performance

The GX saturates the bandwidth of the currently available VRAM. Our design goal—having any drawing operation continuously access the VRAMs at their maximum bandwidth—meant that we had to design each of the three algorithms in the GX from the frame buffer backwards to the CPU. Let's look at what happens when the GX draws a polygon.

The FBC horizontal-fill hardware must process a horizontal span at the rate the VRAMs can accept data. The current board allows a page-mode read or write access to 16 pixels in the VRAM every 129 nanoseconds (or in three 43-ns FBC cycles). The 9-ns difference between this cycle time and the theoretically fastest cycle time of the VRAM (120 ns) allows a margin for manufacturing reliability. Read-modify-write cycles (required by raster operations) take seven cycles (301 ns) for 16 pixels. The cycles occur in the FBC's data path and memory interface

Table 1.
VRAM device and related frame-buffer bandwidths.

VRAM size (bytes)	VRAM configuration (bits)	Page-mode cycle time (ns)	Time required to load each VRAM (ms)	Bandwidth of each VRAM (Mbytes/s)	Number of VRAMs	Total bandwidth into frame buffer (Mbytes/s)
256K	64K × 4	120	8	4	32	128
1M	128K × 8	60	8	16	8	128
1M	256K × 4	60	16	8	8	64
4M	512K × 8	60 *	32	16	2	32
4M	1M × 4	60 *	64	8	2	16

* We've heard talk of implementing a superfast page-mode cycle (25 ns) in future VRAMs.

section shown in Figure 2.

The mask-generation logic must calculate the intersection of the clip window and the edges of the polygon to determine which pixels in the frame buffer are changed at the rate that the memory interface section can access the VRAM. In this case, the worst case object would be a vertical line. Since the horizontal-fill logic can write data every three cycles (129 ns), the mask-generation logic must create a write-enable mask every three cycles.

The functional addressing logic contains the two Bresenham state machines, which must calculate the two x coordinates of the polygon for a given y coordinate. It also contains the clip boundary information. It must calculate and "walk" the polygon's edges at the rate that the mask-generation logic can accept data.

The coordinate-sequencing stage decides which pairs of edges of the polygon are currently being drawn. This bottleneck in firmware-driven graphics accelerators occurs because the decision process proceeds serially. In the FBC combinational logic, which looks at information from the coordinate-staging logic, makes the decision. The coordinate-staging logic creates setup status information based on the vertices of the polygon. The FBC sets up the information in 86 ns (two clock cycles) by comparing the four vertices 56 times with each other and against the clip and clip test windows. Thus the FBC generates 44 bits of comparison information, which is then available for immediate use. Determining whether the FBC must render with a self-intersecting, hidden, or degenerate polygon becomes a simple combinational decoding operation.

The CPU could saturate the FBC if it could read data from main memory and write it to the FBC every 86 ns. The GX's current hosts take between 350 ns and 500 ns to complete read and write cycles for the accelerator. This speed leaves plenty of room for increased performance when the GX runs in faster, future systems.

The CPU has the option of writing the coordinates first to the TEC where they can be transformed and then automatically loaded into the FBC. To avoid becoming the bottleneck, the TEC would have to transform the next set of coordinates in the time it takes to draw an average-size object on the screen (a 10-pixel line or 100-pixel quadrilateral). We've determined the floating-point processing speed requirement to be about 50 Mflops.[13]

When the functionality and performance of all of these logic blocks are put together, the CPU becomes the bottleneck of the graphics system. The GX's performance is directly proportional to the rate the CPU can write data into the GX. We've marketed the GX as being scalable to the CPU's MIPS performance, but it is really scalable to the workstation's I/O performance. Although in general I/O performance has been scaling along with the CPU MIPS and Mflops rates, it is not always in the same proportions. Thus, we can expect GX performance to increase in the future, though it will vary from system to system.

If the GX already runs at the bandwidth of the VRAMs, where will we find the next leap in performance? One improvement could result from the workstation and computer manufacturers' assuming a very active roll in the design of future VRAMs to make sure bandwidth improvements occur. Care must be taken in adding fancy features to VRAMs such as raster operations or filling modes that require information to be stored in the VRAM. In multitasking systems the operating system kernel would have to "understand" how to context switch this information from the VRAM. Other areas for overall system performance improvements could result from moving to the GX those graphics algorithms presently processed in software by the CPU.

Due to the GX design approach, next-generation

Table 2.
Comparing performance in two systems.

Parameters	Sun Microsystems Sparcstation 1GX 4/60GX-8	Silicon Graphics Personal Iris 4D/20	Units
2D vectors	470	80	Kvectors/s
3D vectors	244	80	Kvectors/s
Three-sided, flat-shaded polygon (mesh 50 pixel)	244	16	Ktriangles/s
Four-sided, flat-shaded polygon (independent 10 × 10 pixel)	10	10	Kquadrilaterals
Screen clearance time	8	8	Milliseconds

ASIC techniques can offer little in the area of performance improvements. Adjusting ASICs to higher frequency clocks doesn't mean that the VRAMs can be clocked any faster. The only area to benefit from faster silicon is the transformation engine. Even the TEC's 50-Mflops processing rate is not enough to sustain the rendering hardware at the VRAM bandwidth limits when the GX runs with faster CPUs. The TEC will need to increase proportionally with the rate the CPU can load data into the GX. Beyond this, however, the new submicrometer CMOS processes offer only cost reduction and further integration opportunities.

Comparison

The design approach and concepts embedded in the GX place it in a price/performance class by itself. Table 2 compares performance rates in the Sparcstation 1GX and Silicon Graphic's Personal Iris.[14] Both companies comparably configured these systems at about $15,000.

The 3D vectors and the three-sided polygon meshes both require only one coordinate to be loaded per item drawn. Since the GX uses the same algorithm to draw both a vector and a triangle, the performance rates between these benchmarks are the same. The 3D independent, four-sided polygons require four times the amount of data to be loaded per item than do vectors or triangle meshes. The quadrilaterals are not exactly four times slower than the vectors because the status of the object that is being drawn must be read from the FBC for each object drawn. The status tells the CPU if any exception has occurred such as numerical overflows or full buffers. If no exception is present, the act of reading the FBC initiates the drawing operation. Also, notice that both systems can clear the frame buffer in the same amount of time, which is the same as the bandwidth of the currently available VRAMs (64K × 4 and 128K × 8).

Lowest common denominator

We wanted the GX to bring Sun one step closer to redefining the base functionality of a workstation. Realization of this goal could include the GX in the lowest common denominator (meaning nonoptional, always available on all platforms) of capability between different models of workstations. At first, this may not seem that important a position to attain, but from an application programmer's point of view, the base configuration becomes very important.

Always having graphics acceleration available means that an application can be designed to use display lists and other techniques to exploit the acceleration. An application can be highly tuned if it is to run on only one workstation configuration. If it needs to run on an unaccelerated frame buffer, designers would have to compromise on the features and performance between optimizing for the accelerated and the unaccelerated case. Math-intensive applications evolved the same way, until the floating-point units became standard even on the most cost-sensitive workstations.

With the high levels of integration and the minimal part count of the GX (the entire Sbus-based GX fits on a 6 × 5-inch PC board), somewhat inevitably we will soon see the cost difference between an unaccelerated frame buffer and the GX be reduced to insignificant levels. At that point, the GX could become the graphics workstation's least common denominator.

Software interface

Previous graphics systems containing some type of controller normally communicated data through a display list. This list is simply a well-defined data structure of graphics commands and the coordinates of the objects that the graphics system is to draw. The host CPU would normally create the display list by moving

the data from the user's or application's database into the display list data structure. This data would then be fed to the graphics system through shared memory and first-in, first-out or direct memory access. The graphics system would next transverse the display list and render the object. Users had to buy a dual-processor system to move data around twice as inefficiently.

However, with the GX the CPU becomes an intelligent DMA controller. The CPU reads the data out of the graphics language's display list and writes it into the memory-mapped registers in the GX's ASICs. This design makes the GX very flexible since every graphics language (PHIGS, GKS, or Pixrect) has its own display list formats.

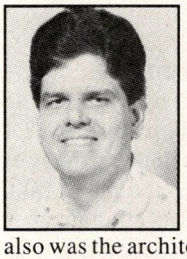
The CPU also makes meaningful decisions part way through the rendering of a picture. For example, if an object moves too far away from the viewer, the CPU could decide against drawing the object and skip to the next one.

GX programmers work in a native environment with high-level languages such as C and debugging tools like dbx on their own workstations. Special simulators, assemblers, microcode, or hardware are unnecessary since the programmers are using the same processor for the graphics language as they are using for driving the GX. High-level graphics programmers still control low-level GX interactions by accessing any and all GX registers. This level of device control had previously only been available to the firmware microcoder on most other graphics subsystems.

The GX accelerates more than vectors and polygons; the most used graphics program is the windowing system itself. The GX provides acceleration for Sun Windows, Sun Tools, X11/News, and Open Look. Users can accelerate applications graphics after porting to one of the graphics interfaces we support, such as Pixrect, GKS, or PHIGS. The GX contains an unaccelerated, dumb frame-buffer interface so that any application that does not run on one of these graphics languages will still work, just slower.

References

1. J.D. Foley and A. van Dam, *Fundamentals of Interactive Computer Graphics,* Addison-Wesley Publishers, Reading, Mass., 1982.
2. W.M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics,* McGraw-Hill Book Company, New York, 1979.
3. C.A. Malachowsky and C.R. Priem, "The GX Graphics Accelerator Architecture," *Sun Tech. J.,* Sun Microsystems, Inc., Mountain View, Calif., Vol. 2, No. 4, Autumn 1989, pp. 85-93.
4. C.A. Malachowsky, "Designing a Low-Cost Scalable Graphics Accelerator," *High-Performance Systems,* Vol. X, No. 5, May 1989, pp. 68-78.
5. J.E. Bresenham, "An Incremental Algorithm for Digital Plotting," *Proc. ACM Nat'l Conf.,* Aug. 1963.
6. J.E. Bresenham, "Algorithm for Computer Control of a Digital Plotter," *IBM System J.,* Journal 4, No. 1, July 1965, pp. 25-30.
7. J.E. Bresenham, "Run Length Slices for Incremental Lines," *IBM Technical Disclosure Bulletin 22-8B,* Jan. 1980, pp. 3744-3747.
8. J.E. Bresenham, "Incremental Line Compaction," *The Computer J.,* Vol. 25, No. 1, 1982, pp. 116-120.
9. J.F. Blinn and M.E. Newell, "Clipping Using Homogeneous Coordinates," *ACM Computer Graphics,* Vol. 12, No. 3, Aug. 1978, pp. 197-203.
10. J.J.F. Cavanagh, "Digital Computer Arithmetic: Design and Implementation," McGraw-Hill, 1984, pp. 278-284.
11. S. Waser and M.J. Flynn, *Introduction to Arithmetic for Digital Systems Designers,* CBS College Publishing, New York, 1982, pp. 196-199.
12. *ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic,* IEEE Computer Society Press, Los Alamitos, Calif., 1985.
13. K. Akeley and T. Jermoluk, "High-Performance Polygon Rendering," *ACM Computer Graphics,* Vol. 22, No. 4, Aug. 1988, pp. 239-246.
14. Silicon Graphics Inc., *Personal IRIS Specification, (Preliminary),* Mountain View, Calif., Oct. 1988, p. 4.



Curtis R. Priem, a staff engineer at Sun Microsystems, Inc., is the architect of the GX graphics accelerator products. He also designed the transformation engine section of the GX's TEC. Prior to joining Sun, he worked at Genrad, where he developed the Hichip hardware modeler. He also was the architect and principal designer of IBM's Professional Graphics Adapter while at Vermont Microsystems Inc.

Priem holds a BS degree in computer systems engineering from Rensselaer Polytechnic Institute.

Direct questions concerning this article to the author at Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, CA 94043, or by electronic mail at curtis@sun.com.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 168 **Medium** 169 **High** 170